



NISHANT SHINDE

HMS should be capable to recognize already registered patients and user roles.

1. Write necessary queries to register new user roles and personas

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the following query:

```
1 INSERT INTO Users (UserID, Username, Password, Role)
2 VALUES
3 (4, 'doctor2', 'docpass2', 'doctor'),
4 (5, 'doctor3', 'docpass3', 'doctor'),
5 (6, 'receptionist2', 'recpass2', 'receptionist');
6
```

The Output tab shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	23.13.32	CREATE TRIGGER CheckInsuranceLimit AFTER INSERT ON MedicalRecords FOR EACH ROW BEGIN	DECLA... Error Code: 1359. Trigger already exists	0.015 sec
2	23.36.04	INSERT INTO Users (UserID, Username, Password, Role) VALUES (4, 'doctor2', 'docpass2', 'doctor'), (5, 'doctor3', 'docpass3', 'doctor'), (6, 'receptionist2', 'recpass2', 'receptionist');	3 row(s) affected. Records: 3 Duplicates: 0 Warnings: 0	0.015 sec

2. Write necessary queries to add to the list of diagnosis of the patient tagged by date.

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the following queries:

```
1 INSERT INTO medicalrecords (RecordID, PatientID, DoctorID, Date, Diagnosis, Prescription, Notes)
2 VALUES
3 (2, 1, 1, '2024-01-29', 'cough', 'Med3, Med2', 'Drink hot water')
4
5 SELECT * FROM hms_sankey.medicalrecords;
6
```

The Output tab shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	10.54.16	INSERT INTO medicalrecords (RecordID, PatientID, DoctorID, Date, Diagnosis, Prescription, Notes) VALUES (2, 1, 1, '2024-01-29', 'cough', 'Med3, Med2', 'Drink hot water');	Error Code: 1146. Table 'hms_sankey.insuranceLimits' doesn't exist	0.016 sec
2	10.55.37	CREATE TABLE InsuranceLimits (PatientID INT PRIMARY KEY, Limit INT)	0 row(s) affected	0.016 sec
3	10.55.48	CREATE TRIGGER CheckInsuranceLimit AFTER INSERT ON MedicalRecords FOR EACH ROW BEGIN	... Error Code: 1359. Trigger already exists	0.000 sec
4	10.56.15	INSERT INTO medicalrecords (RecordID, PatientID, DoctorID, Date, Diagnosis, Prescription, Notes) VALUES (2, 1, 1, '2024-01-29', 'cough', 'Med3, Med2', 'Drink hot water');	1 row(s) affected	0.000 sec
5	10.56.33	SELECT * FROM hms_sankey.medicalrecords LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
6	10.57.08	SELECT * FROM hms_sankey.medicalrecords LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

The Result Grid shows the following data:

RecordID	PatientID	DoctorID	Date	Diagnosis	Prescription	Notes
1	1	2	2024-01-29	Common cold	Med1, Med2	Patient needs rest and hydration.
2	1	1	2024-01-29	cough	Med3, Med2	Drink hot water

3. Write necessary queries to fetch required details of a particular patient.

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the following query:

```
1 SELECT * FROM Patients WHERE PatientID = 1;
```

The Result Grid displays the following data:

PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address
1	John	Doe	1990-05-15	M	123-456-7890	123 Main St

The Output tab shows the execution log with the following actions:

#	Time	Action	Message	Duration / Fetch
1	10:54:16	INSERT INTO medicalrecords (RecordID, PatientID, DoctorID, Date, Diagnosis, Prescription, Notes) VALUES ...	Error Code: 1146. Table 'hms_sankey.insuranceLimits' doesn't exist	0.016 sec
2	10:55:37	CREATE TABLE insuranceLimits (PatientID INT PRIMARY KEY, 'Limit' INT)	0 row(s) affected	0.016 sec
3	10:55:48	CREATE TRIGGER CheckInsuranceLimit AFTER INSERT ON MedicalRecords FOR EACH ROW BEGIN -- A...	Error Code: 1359. Trigger already exists	0.000 sec
4	10:56:15	INSERT INTO medicalrecords (RecordID, PatientID, DoctorID, Date, Diagnosis, Prescription, Notes) VALUES ...	1 row(s) affected	0.000 sec
5	10:56:33	SELECT * FROM hms_sankey.medicalrecords LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
6	10:57:08	SELECT * FROM hms_sankey.medicalrecords LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
7	10:58:36	SELECT * FROM Patients WHERE PatientID = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

4. Write necessary queries to prepare bill for the patient at the end of checkout.

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the following queries:

```
1 INSERT INTO Billing (BillID, PatientID, DoctorID, AppointmentID, Date, TotalAmount, PaymentStatus)
2 VALUES
3 (2, 1, 2, 1, '2024-01-29', 150.00, 'Pending');
4
5 select * from billings;
```

The Result Grid displays the following data:

BillID	PatientID	DoctorID	AppointmentID	Date	TotalAmount	PaymentStatus
1	1	2	1	2024-01-29	100	Pending
2	1	2	1	2024-01-29	150	Pending

The Output tab shows the execution log with the following actions:

#	Time	Action	Message	Duration / Fetch
3	10:55:48	CREATE TRIGGER CheckInsuranceLimit AFTER INSERT ON MedicalRecords FOR EACH ROW BEGIN -- A...	Error Code: 1359. Trigger already exists	0.000 sec
4	10:56:15	INSERT INTO medicalrecords (RecordID, PatientID, DoctorID, Date, Diagnosis, Prescription, Notes) VALUES ...	1 row(s) affected	0.000 sec
5	10:56:33	SELECT * FROM hms_sankey.medicalrecords LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
6	10:57:08	SELECT * FROM hms_sankey.medicalrecords LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
7	10:58:36	SELECT * FROM Patients WHERE PatientID = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
8	10:59:15	INSERT INTO Billing (BillID, PatientID, DoctorID, AppointmentID, Date, TotalAmount, PaymentStatus) VALUE...	Error Code: 1062. Duplicate entry '1' for key 'billing.PRIMARY'	0.000 sec
9	10:59:20	INSERT INTO Billing (BillID, PatientID, DoctorID, AppointmentID, Date, TotalAmount, PaymentStatus) VALUE...	Error Code: 1062. Duplicate entry '1' for key 'billing.PRIMARY'	0.000 sec
10	10:59:36	INSERT INTO Billing (BillID, PatientID, DoctorID, AppointmentID, Date, TotalAmount, PaymentStatus) VALUE...	1 row(s) affected	0.000 sec
11	11:00:01	select * from billings LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

5. Write necessary queries to fetch and show data from various related tables (Joins)

The screenshot shows the MySQL Workbench interface. The SQL editor contains a query that joins the `Appointments` table with the `Patients` and `Doctors` tables. The query is as follows:

```

1 SELECT A.AppointmentID, P.FirstName AS PatientFirstName, P.LastName AS PatientLastName,
2       D.FirstName AS DoctorFirstName, D.LastName AS DoctorLastName,
3       A.AppointmentDate, A.AppointmentTime, A.Status
4 FROM Appointments A
5 JOIN Patients P ON A.PatientID = P.PatientID
6 JOIN Doctors D ON A.DoctorID = D.DoctorID
7 WHERE A.AppointmentID = 1;

```

The Result Grid shows the output of the query:

AppointmentID	PatientFirstName	PatientLastName	DoctorFirstName	DoctorLastName	AppointmentDate	AppointmentTime	Status
1	John	Doe	Dr. Sarah	Jones	2024-02-01	10:00:00	Scheduled

The Action Output pane shows the execution log, including the query execution and the resulting data rows.

6. Optimize repeated read operations using views/materialized views.

The screenshot shows the MySQL Workbench interface. The SQL editor contains a query that creates a view named `PatientAppointmentsView`. The query is as follows:

```

1 CREATE VIEW PatientAppointmentsView AS
2 SELECT
3     A.AppointmentID,
4     P.FirstName AS PatientFirstName,
5     P.LastName AS PatientLastName,
6     D.FirstName AS DoctorFirstName,
7     D.LastName AS DoctorLastName,
8     A.AppointmentDate,
9     A.AppointmentTime,
10    A.Status
11 FROM
12     Appointments A
13 JOIN Patients P ON A.PatientID = P.PatientID
14 JOIN Doctors D ON A.DoctorID = D.DoctorID;

```

The Action Output pane shows the execution log, including the view creation and the resulting data rows.

7. Optimize read operations using indexing wherever required. (Create index on at least 1 table)

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following queries:

```
1 SELECT * FROM appointments WHERE AppointmentDate = '2024-02-01';
2
```

The Result Grid shows the following data:

AppointmentID	PatientID	DoctorID	AppointmentDate	AppointmentTime	Status
1	1	2	2024-02-01	10:00:00	Scheduled

The Action Output pane shows the execution of the following queries:

```
10 10:59:36 INSERT INTO Billing (BillID, PatientID, DoctorID, AppointmentID, Date, TotalAmount, PaymentStatus) VALUE... 1 row(s) affected 0.000 sec
11 11:00:01 select * from billing LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
12 11:00:42 SELECT A.AppointmentID, P.FirstName AS PatientFirstName, P.LastName AS PatientLastName, D.Fir... 1 row(s) returned 0.000 sec / 0.000 sec
13 11:02:38 CREATE INDEX idx_AppointmentDate ON appointments(AppointmentDate) Error Code: 1061. Duplicate key name 'idx_AppointmentDate' 0.015 sec
14 11:02:44 CREATE INDEX idx2_AppointmentDate ON appointments(AppointmentDate) 0 row(s) affected, 1 warning(s): 1831 Duplicate index 'idx2_AppointmentDate' defined on the table 'hms_sankey...' 0.047 sec
15 11:03:32 SELECT * FROM appointments WHERE AppointmentDate = '2024-02-01' LIMIT 0, 1000 1 row(s) returned 0.000 sec / 0.000 sec
16 11:03:55 SELECT * FROM hms_sankey.patientappointmentsview LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
17 11:04:24 SELECT * FROM hms_sankey.appointments LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
18 11:05:14 SELECT * FROM hms_sankey.appointments LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
```

8. Try optimizing bill generation using stored procedures.

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following queries:

```
1 DELIMITER //
2 CREATE PROCEDURE CalculateTotalAmount(IN appointmentID INT)
3 BEGIN
4 DECLARE total DECIMAL(10, 2);
5 SET total = 100.00;
6 UPDATE Billing SET TotalAmount = total WHERE AppointmentID = appointmentID;
7 END //
8 DELIMITER ;
9
10 CALL CalculateTotalAmount(1);
```

The Result Grid shows the following data:

TotalAmount
100

The Action Output pane shows the execution of the following queries:

```
16 11:03:55 SELECT * FROM hms_sankey.patientappointmentsview LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
17 11:04:24 SELECT * FROM hms_sankey.appointments LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
18 11:05:14 SELECT * FROM hms_sankey.appointments LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
19 11:09:18 DROP PROCEDURE IF EXISTS CalculateTotalAmount 0 row(s) affected 0.016 sec
20 11:11:02 CREATE PROCEDURE CalculateTotalAmount(IN appointmentID INT) BEGIN DECLARE total DECIMAL(10... 0.015 sec
21 11:11:12 CALL CalculateTotalAmount(1) 1 row(s) affected 0.000 sec
22 11:11:39 CALL CalculateTotalAmount(2) 0 row(s) affected 0.000 sec
23 11:11:44 CALL CalculateTotalAmount(2) 0 row(s) affected 0.000 sec
24 11:12:53 SELECT TotalAmount FROM Billing WHERE AppointmentID = 1 LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
```

9. Add necessary triggers to indicate when patients medical insurance limit has expired.

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view of the database structure, including tables like 'appointments', 'billing', 'doctors', 'insuranceLimits', 'medicalRecords', 'patients', and 'users'. The main editor window shows a SQL script for creating a trigger named 'CheckInsuranceLimit'. The script is as follows:

```
1 DELIMITER //
2 CREATE TRIGGER CheckInsuranceLimit
3 AFTER INSERT ON MedicalRecords
4 FOR EACH ROW
5 BEGIN
6     DECLARE insuranceLimit INT;
7
8     SELECT 'Limit' INTO insuranceLimit FROM InsuranceLimits WHERE PatientID = NEW.PatientID;
9
10    IF insuranceLimit <= 0 THEN
11
12        UPDATE Patients SET InsuranceStatus = 'Expired' WHERE PatientID = NEW.PatientID;
13    END IF;
14 END //
15 DELIMITER ;
```

The right sidebar shows the 'SQL-Additions' panel with a message: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.'

The bottom panel shows the 'Output' window with the 'Action Output' tab selected. It displays a table of execution results:

#	Time	Action	Message	Duration / Fetch
22	11:11:39	CALL CalculateTotalAmount(2)	0 row(s) affected	0.000 sec
23	11:11:44	CALL CalculateTotalAmount(2)	0 row(s) affected	0.000 sec
24	11:12:53	SELECT TotalAmount FROM Billing WHERE AppointmentID = 1 LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
25	11:17:48	EXPLAIN SELECT TotalAmount FROM Billing WHERE AppointmentID = 1	OK	0.000 sec
26	11:17:48	EXPLAIN FORMAT=JSON SELECT TotalAmount FROM Billing WHERE AppointmentID = 1	OK	0.000 sec
27	11:18:10	CREATE TRIGGER CheckInsuranceLimit AFTER INSERT ON MedicalRecords FOR EACH ROW BEGIN	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se...	0.000 sec
28	11:19:11	CREATE TRIGGER CheckInsuranceLimit AFTER INSERT ON MedicalRecords FOR EACH ROW BEGIN	Error Code: 1359. Trigger already exists	0.000 sec
29	11:19:51	drop trigger if exists CheckInsuranceLimit	0 row(s) affected	0.000 sec
30	11:19:57	CREATE TRIGGER CheckInsuranceLimit AFTER INSERT ON MedicalRecords FOR EACH ROW BEGIN	0 row(s) affected	0.016 sec

The bottom status bar shows the system clock at 11:21 AM on 04/02/2024, along with other system icons.

Question 2

Write a report on your understanding of Rendering and Design Patterns. Mention and elaborate where a particular Rendering pattern is applicable and is well suited for which use case.

Rendering Patterns

→

Rendering Patterns

1. **Client-Side Rendering:**
2. **Incremental Static Generation:**
3. **Stepwise Hydration:**
4. **Selective Hydration:**
5. **React Server Components:**
6. **Server-side rendering:**
7. **Static Rendering:**
8. **Server-side streaming rendering:**

Client-Side Rendering:

When a website uses client-side rendering, your browser is responsible for rendering the web page for you. It's fast at first, but can get slower as you progress because it depends on your browser to do so much work.

Incremental Static Generation:

Think of Incremental Static Generation as a fast and reliable website that can show you new content instantly. It prepares pages in advance but updates them dynamically as needed.

Stepwise Hydration:

It's like creating a website step by step. It starts with the basics and then adds more interactive features to make pages load faster and get better over time.

Selective Hydration:

Selective Hydration refers to the skillful use of the web. It saves time and speeds up the process by loading beautiful interactive content only when you need it.

React Server Components:

React server components are like the building blocks of a website. It is designed to make large websites run more efficiently by doing some of the heavy lifting on the server.

Server-side rendering:

Server-side rendering is when the server creates a web page and sends it to your browser. It makes homepages load faster and helps search engines, but it can increase the server's workload.

Static Rendering:

Static rendering is like having a ready-made web page. These are pre-made and can be shipped quickly as their stock is waiting to be discovered.

Server-side streaming rendering:

Server-side streaming rendering will present various parts of the web page to you as they become available. This way your browser can start showing your page immediately and make it faster.

Design Patterns

1. **Compound pattern:**
2. **HOC Pattern (Higher Order Component):**
3. **Hooks Model:**
4. **Container/Presentation Pattern:**
5. **Render Props Pattern:**

1.Compound pattern:

A compound pattern is a design that combines several designs to solve complex problems.

They are dependent on each other with shared state and logic

It leverages the power of various models to solve complex software architecture problems.

2.HOC Pattern (Higher Order Component):

Higher Order Component (HOC) pattern is a widely used design pattern in React.js.

This helps us to reuse patterns throughout our applications

HOC is component that receives another component, also It uses logic that are used to pass as a parameters, after applying logic it passes and return the element with additional logic

3.Hooks Model:

The Hooks model, introduced in React 16.8, revolutionizes state management and side effect management in functional components.

Hooks like `useState` and `useEffect` allow developers to handle state events and lifecycle events on object, eliminating the need for multiple class objects.

The Hooks pattern increases the readability and ease of using code for developing React applications.

4.Container/Presentation Pattern:

Container/Presentation Pattern is a design pattern that is divided into two main types: container and presentation.

Containers manage state and logic, while presentation elements focus solely on rendering UI elements. This separation increases the clarity and security of the code and makes it easier to test and reuse in front-end development.

5.Render Props Pattern:

Render Props Pattern is a design pattern in React that involves using functions as props for objects.

This function, called render prop, allows the element to share code or state with its children.

Render prop mode facilitates integration and code reuse by customizing prop to specific need, thus promoting flexibility and extending code structure.